Today’s Lecture

- Cleanup – Lecture 3
  - TCP Socket Programming
  - Project 1 Foundation
- Layer Magic
  - Start with Physical Layer
  - Chapter 2.1-2.3

Socket Addresses

```c
(1)
struct sockaddr {
    sa_family_t sa_family; /* address family */
    char sa_data[14];    /* addressing information */
};

(2)
struct sockaddr_in {
    short sin_family;     /* AF_INET */
    u_short sin_port;     /* network byte order */
    struct in_addr sin_addr; /* network address */
};
```

Two options
(1) is most common, (2) is actually the recommended approach
sendto

ssize_t sendto(int socket,
    const void *message,
    size_t length,
    int flags,
    const struct sockaddr *dest_addr,
    socklen_t dest_len);

Socket
Pointer to a pre-allocated buffer
Size of the buffer (to avoid overflow)
Flags (usually 0)
Address struct – where the packet came from
Length of the address

In-Class Derivation

• Write a small snippet of code to send a packet via UDP

Where next?

• UDP – protocol is up to programmer
  – How many messages?
  – Who talks when?
  – What type of message?

Homework 2: Write a simple UDP client
  Read a small text file
  Send to the server
  Read / display the response
TCP vs. UDP

- UDP – User Datagram Protocol
  - Lightweight
  - Very flexible
  - Largely user-defined in terms of behavior
    - How fast to send?
    - What happens when messages are lost?
- TCP – Transmission Control Protocol
  - Heavyweight
  - Reliable
  - “Good” network citizen

TCP

- Reliable, in-order data delivery
  - Hands off approach to network programming
  - Most common protocol on network
- Concept of a connection / stream
  - Set up a “tube” to send data back / forth
  - Server listens for connections
    - Chooses whom to accept
  - Client connects to the server
    - Three-way handshake (TWH)
      - SYN, SYN-ACK, ACK (flags in TCP)

Revisiting TCP – Server Side

[Diagram of TCP connection setup and functions]

TCP Only
TCP, UDP
listen()

```c
int listen(int s, int backlog);
```

- Only for stream sockets.
- Socket being listened to can’t be used for client.
- `listen()` does not wait for connections, but needed to receive connection attempts.
- Completes 3-way handshake.

- Can queue up connections
  - Specify how many to backlog
  - Completed handshake
- What happens when it is full?
  - SYN requests are silently dropped
- Denial of Service (DoS)
  - SYN flood – overload queues

TCP Preview

![TCP Preview Diagram](image)
accept()

int accept(int s,
           struct sockaddr *addr,
           int *addrlen);

* By connection-oriented server, after listen()
* Cannot preview connection
  – accept and close
* Returns a new socket
* Returns address of client

Client socket, how the server talks to a specific client

Sending/Receiving

* TCP
  – read/write: send/receive, no explicit address
  – send/receive, flags
* UDP
  – sendto: specify destination explicitly
  – recvfrom: also receive address of peer
  – sendmsg: msghdr data structure, scatter/gather
  – recvmsg: msghdr data structure, scatter/gather

Example: TCP Client

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
   int s, n;
   struct sockaddr_in sin;
   char msg[80] = "Hello, World!
   if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      perror("socket"); return(-1);
   }
   // Further code...
}
```
Example: TCP Client

```c
sin.sin_family = AF_INET;
sin.sin_port = htons(atoi(argv[2]));
sin_addr.s_addr = inet_addr(argv[1]);
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("connect"); return -1;
}
if (write(s, msg, strlen(msg)+1) < 0) {
    perror("write"); return -1;
}
if ((n = read(s, msg, sizeof(msg))) < 0) {
    perror("read"); return -1;
}
printf("%d bytes: ", n);
if (close(s) < 0) {
    perror("close"); return -1;
}
return 0;
```

Single-Threaded Server
(Pseudocode)

```c
int server;
server = initialize();
while (1) {
    wait_for_request(server);
    read_request;
    send_answer;
}
```

Example: TCP Server

```c
main(int argc, char *argv[]) {
    int s, t, n;
    struct sockaddr_in sin;
    char *r;
    char buf[100];
    int sinlen;
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket"); return -1;
    }
    sin.sin_family = AF_INET;
    sin.sin_port = htons(13333);
    sin.sin_addr.s_addr = INADDR_ANY;
    if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        perror("bind"); return -1;
    }
    if (listen(s, 5) < 0) {
        perror("listen"); return -1;
    }
    return 0;
}
```
Example: TCP Server

```c
for (;;) {
    sinlen = sizeof(sin);
    if ((t = accept(s, (struct sockaddr *) &sin, &sinlen)) < 0) {
        perror("accept"); return -1;
    }
    if (read(t, buf, 100) < 0) {
        perror ("read"); return -1;
        }
    r = gettime();
    if (write(t, r, strlen(r)) < 0) {
        perror("write"); return -1;
    }
    if (close(t) < 0) {
        perror("close"); return -1;
    }
}
if (close(s) < 0) {
    perror("close"); return -1;
}
```

Other Socket Functions

- The socket API contains a variety of support functions
  - getpeername
  - gethostname
  - setsockopt
  - getsockopt
  - gethostbyname
  - gethostbyaddr

What About Threaded Servers?

- The socket API works well with concurrent servers
- Implementations of the socket API adhere to the following inheritance principle:
  - each new thread that is created inherits a copy of all open sockets from the thread that created it
  - the socket implementation uses a reference count mechanism to control each socket
  - when a socket is first created
    * the system sets the socket's reference count to 1
    * and the socket exists as long as the reference count remains positive
  - when a program creates an additional thread
    * the thread inherits a pointer to each open socket the program owns
    * and the system increments the reference count of each socket by 1
  - when a thread calls close
    * the system decrements the reference count for the socket
    * if the reference count has reached zero, the socket is removed
Why Threads?

- How many people access a server?
  - One at a time, wait until they finish
  - Multiple in, multiple out
- Threads
  - Magic that allows us to handle multiple clients
  - Create one thread / client socket

Bringing it Around

- Why cover this now?
  - Allow you to program now
  - Socket programming
    - Most of you will end up doing
    - More practice = better
  - Incremental projects
    - UDP Client
    - TCP Client
    - TCP Server
    - Protocol Analysis, etc.

Direct Link Networks

Outline
- Building Blocks
- Encoding

Outline
- Application
- Transport
- Network
- Data
- Physical
Nodes

- Network adapter
  - Interfaces
    - eth0, eth1
- Device driver
  - How to talk to device?
- Memory bottleneck
  - How fast can I read / write?

Links

- Electromagnetic waves traveling at speed of light
- Frequency (Hertz, Hz) and wavelength (meters)

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Waveband</th>
</tr>
</thead>
<tbody>
<tr>
<td>10¹⁵ Hz</td>
<td>Gamma ray</td>
</tr>
<tr>
<td>10¹¹ Hz</td>
<td>X ray</td>
</tr>
<tr>
<td>10¹⁰ Hz</td>
<td>Ultraviolet</td>
</tr>
<tr>
<td>10⁶ Hz</td>
<td>Infrared</td>
</tr>
<tr>
<td>10⁴ Hz</td>
<td>Microwave</td>
</tr>
<tr>
<td>10³ Hz</td>
<td>Radio</td>
</tr>
</tbody>
</table>

- Cables:
  - Category 5 twisted pair: 10-100 Mbps, 100m
  - Thin-net coax: 10-100 Mbps, 500m
  - Thick-net coax: 10-100 Mbps, 200m
  - Multimode fiber: 100 Mbps, 2km
  - Single-mode fiber: 100-2400 Mbps, 40km

- Leased lines:
  - DS1: 1.544 Mbps
  - DS3: 44.736 Mbps
  - STS-1: 51.840 Mbps
  - STS-3: 155.250 Mbps
  - STS-12: 622.080 Mbps
  - STS-48: 2.48832 Gbps
  - STS-192: 9.95328 Gbps
Categories of Twisted Pair Cables

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Data Rate (in Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT 1</td>
<td>Unshielded twisted pair used for telephones</td>
<td>0.1</td>
</tr>
<tr>
<td>CAT 2</td>
<td>Unshielded twisted pair used for T1 data</td>
<td>2</td>
</tr>
<tr>
<td>CAT 3</td>
<td>Improved CAT2 used for computer networks</td>
<td>50</td>
</tr>
<tr>
<td>CAT 4</td>
<td>Improved CAT3 used for Token Ring networks</td>
<td>20</td>
</tr>
<tr>
<td>CAT 5</td>
<td>Unshielded twisted pair used for networks</td>
<td>100</td>
</tr>
<tr>
<td>CAT 5E</td>
<td>Extended CAT5 for more noise immunity</td>
<td>128</td>
</tr>
<tr>
<td>CAT 6</td>
<td>Unshielded twisted pair tested for 200 Mbps</td>
<td>200</td>
</tr>
<tr>
<td>CAT 7</td>
<td>Shielded twisted pair with a foil shield around the entire cable plus a shield around each twisted pair</td>
<td>500</td>
</tr>
</tbody>
</table>

Figure 7.4: Twisted pair wiring categories and a description of each.
Optical Fiber

- Each fiber consists of a thin strand of glass or transparent plastic encased in a plastic cover
  - An optical fiber is used for communication in a single direction
  - One end of the fiber connects to a laser or LED used to transmit light
  - The other end of the fiber connects to a photosensitive device used to detect incoming light

Links

- Last-mile links:
  - POTS 28.8-56 Kbps
  - ISDN 64-128 Kbps
  - xDSL 16 Kbps-55.2 Mbps
  - CATV 20-40 Mbps
  - 1.554-8.448 Mbps
  - 16-640 Kbps

- VDSL at 12.96-55.2 Mbps
- 100-1100 feet of copper
- Over 1000 feet of copper
Example: ADSL

- ADSL is the most widely deployed variant
  - and the one that most residential customers use
- ADSL uses FDM to divide the bandwidth of the local loop into three regions
  - one of the regions corresponds to traditional analog phone service, which is known as Plain Old Telephone Service (POTS)
  - and two regions provide data communication

Links

- Wireless links
  - AMPS: Advanced Mobile Phone Systems
  - PCS: Personal Communication Services
  - GSM: Global System for Mobile Communication
  - infrared: 850-950 nm, 1 Mbps, 10m
  - HIPERLAN: High Performance European Radio LAN
  - IEEE 802.11
  - Bluetooth
  - WiMax

Encoding

- Signals propagate over a physical medium
  - modulate electromagnetic waves
    - e.g., vary voltage
- Encode binary data onto signals
  - e.g., 0 as low signal and 1 as high signal
    - known as Non-Return to zero (NRZ)
Problem: Consecutive 1s or 0s

- Low signal (0) may be interpreted as no signal
- Long strings of 0s or 1s lead to baseline wander
- Unable to recover clock

Alternative Encodings

- Non-return to Zero Inverted (NRZI)
  - make a transition from current signal to encode a one;
    stay at current signal to encode a zero
  - solves the problem of consecutive ones

- Manchester
  - transmit XOR of the NRZ encoded data and the clock
  - only 50% efficient (bit rate = 1/2 baud rate)

Encodings (cont)

- 4B/5B
  - every 4 bits of data encoded in a 5-bit code
  - 5-bit codes selected to have no more than one leading 0
    and no more than two trailing 0s
  - thus, never get more than three consecutive 0s
  - resulting 5-bit codes are transmitted using NRZI
  - achieves 80% efficiency
Encodings (cont)

Manchester Encoding

4B/5B
Framing

- Break sequence of bits into a frame
- Typically implemented by network adaptor

![Diagram of framing](image)

Figure 9.7: Illustration of framing on a synchronous transmission system

Approaches

- Sentinel-based
  - start frame with special pattern: 01111110

Approaches

- Counter-based
  - include payload length in header
  - e.g., DDCMP
  - problem: count field corrupted
  - solution: catch when CRC fails
Approaches

- Bit-oriented: HDLC
  - 01111110 for beginning and end, also sent during idle times for synchronization
  - Bit stuffing: when 5 consecutive 1s have been transmitted, sender inserts 0

Approaches

- Clock-based (SONET)
  - Each frame is 125μs long
  - e.g., SONET: Synchronous Optical Network
  - STS-n (STS-1 ~ 51.84 Mbps)